5.3 Web automation

Cucumber id often used with Selenium WebDriver to write expressive automation tests.

Consider the example available in B. Garcia's repository concerning the integration of Cucumber and Selenium [junit5-cucumber-selenium] and implement it.
Adapt from this example to create a test scenario related to the "BlazeDemo" application (recall Lab 4, exercise 2).

Lab 6 Static Code analysis (with SonarQube)

Context and key points

Key Points
Static code quality can assess a code base to produce quality metrics. These metrics are based on the occurrence of known "bed smells" and weaknesses. In this kind of analysis, the solution is not deployed, nor the code is executed (thus the name static analysis).
Static code analysis can be run locally using a "linter" and modern IDE will typically include support for code inspection. But it would be even more import to implement code analysis in the team develop infrastructure, i.e., at the continuous integration pipeline, using specialized services.
Key code quality measures include the occurrence of problems likely to produce errors, vulnerabilities (security/reliability concerns) and code smells (bad/poor practice or coding style); coverage (ratio tested/total); and code complexity assessment.
The estimated effort to correct the vulnerabilities is called the **technical debt**. Every software quality engineer needs tools to obtain realistic information on the technical debt.

Explore

— public projects on Sonar cloud that you can browse and learn.

6.1 Local analysis

a) Copy a Maven-based, Java application project to use (with tests passing). You may reuse one from previous labs, for example, the Euromillions from Lab 1.2.

b) Prepare a local instance of SonarQube server (using the Docker image). For this lab, you do not need to configure a production database (an embedded database is used by default; for a production scenario, a more demanding configuration is required).

   Note1: you may get a **conflict on port 9000** in your host, as it is also commonly picked for other services (e.g.: Portainer); pick another, if needed.

   Note2: there were known problems to run the provided docker image on Mac M1 (Apple silicon). Consider the following alternatives: use the .zip file version; use an alternative docker image.

c) Confirm that you can access the Sonar dashboard (default : http://127.0.0.1:9000) and change the default credentials (admin / admin).

d) Complete the "Analyzing a Project steps": create a project "manually"; set for "local analysis"; generate a named token. **Take note of the generated user token**! You will need it later several times.

e) Include the Sonar Maven plugin in your POM.

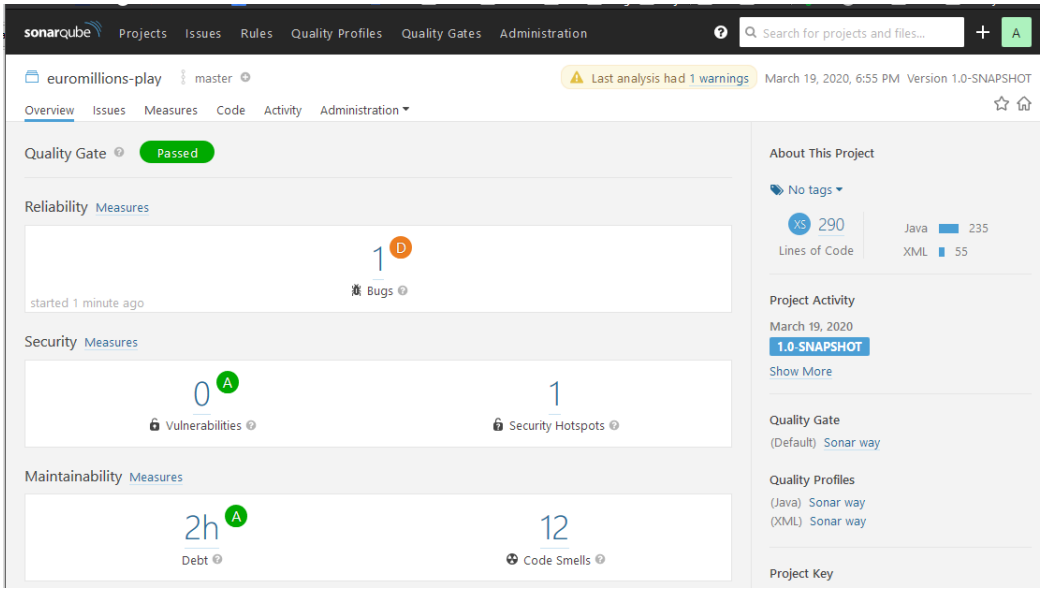
From the command line, run the code analysis (highlighted parts should be changed as needed):

```
$ mvn verify sonar:sonar -Dsonar.host.url=http://localhost:9009
-Dsonar.projectKey=lab6_1 -Dsonar.login=053bd3d423525e6df97b6bfd06b8a7ecd5bb7e
```

Note: optionally, you can save part of the Sonar configuration as "global settings".

f) Confirm that Sonar analysis was executed. Access the SonarQube dashboard (default : http://127.0.0.1:9000).

Has your project passed the defined quality gate? Elaborate your answer (in **Readme** document/markdown file, along with the code project).



g) Explore the analysis results and complete with a few sample issues, as applicable. (Place your response in a **Readme** file, either html/md/pdf…).

Issue	**Problem description**	**How to solve**
Bug	…	…
Vulnerability		
Code smell (major)		

6.2 Technical debt (Cars)

Make a copy of the "Cars" project from Lab #3.2.
Be sure you are using a project with JUnit **tests implemented and passing**.

a) Analyze this project with SonarQube. Remember to create a new project in your Sonar instance and get a token.

Take note of the **technical debt found**. Explain what this value means.

Document the analysis findings with a screenshot (of the sonar dashboard for this project).

b) Analyze the reported problems and be sure to **correct the severe** code smells reported (critical and major).

Note: if you used the Entity data type as parameter in the API methods, you will likely get the vulnerability "Persistent entities should not be used as arguments".

c) Code coverage reports requires the Jacoco plugin. Be sure to use a project with unit tests and configured code coverage (e.g.: add the jacoco plugin to maven and update the plugin version).

d) Run the static analysis and observe/explore the coverage values on the SonarQube dashboard.

How many lines are "not covered"? And how many conditions?

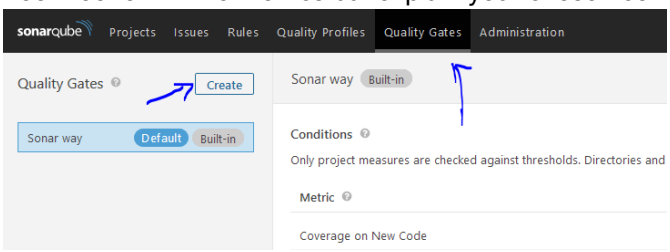
6.3 Custom QG

For this exercise, it would appropriate to use **a larger project**. Consider using your (group) project from IES[5]. Alternatively, you may get an open-source project (maven-based, Java project). Otherwise, continue with the project from previous exercise.

Note: **do not** to submit this project to your TQS personal Git repo! Focus on providing evidence that you complete the tasks and discuss the outcomes in a Readme.md file.

a) If possible, collaborate with other colleagues to define a custom quality gate to this project (especially if you are using the IES project, try to work with the former IES team…).

Feel free to mix the metrics but explain your chosen configuration.



b) Add an increment to the source code. You may try to introduce some "bad smells"; in fact, try to break the quality gate.

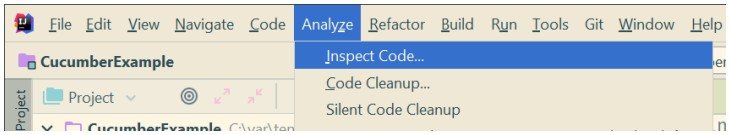
Run the analysis and analyze the results.

6.4 IDE

[Optional. No submission required.]
In the previous tasks, we assume the static code analysis as a service, likely to be integrated at the Continuous Integration pipeline. You will also benefit from having code inspection as you develop, automatically integrated in the IDE.

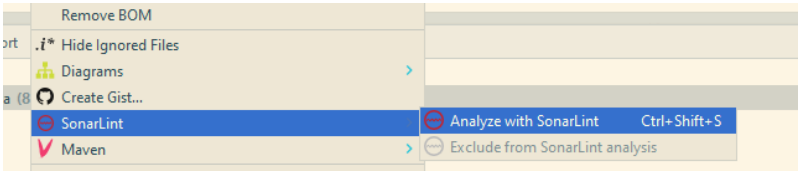
Note that:

a) IntelliJ (and most IDE) already integrates a code inspection tool. If you have not used before, try it.



b) The Sonar Qube analysis can be integrated in IntelliJ through the SonarLint plug-in.

[5] More specifically, the backend/API subproject, if applicable.


Remove BOM
Hide Ignored Files
Diagrams
Create Gist...
SonarLint
Maven
Analyze with SonarLint Ctrl+Shift+S
Exclude from SonarLint analysis